Information Modelling for Configurable Components

Jørgensen, Kaj Asbjørn

Published in:
Proceedings of the Sixth International Conference on Engineering Computational Technology

Publication date:
2008

Document Version
Accepted author manuscript, peer reviewed version

Presented at the ECT2008 conference in Athens.

AaU – Department of Production

2008-06-09

Information Modelling for Configurable Components

Kaj A. Jørgensen

Aalborg University, Dept. of Production, Denmark

kaj@production.aau.dk

Abstract

In this paper, a rather simple approach for modelling of configurable product components is presented. This approach is based on the theory of general systems and outlined in combination with the abstraction mechanisms classification and composition together with object-oriented analysis and design. Throughout the presentation, a previously developed generic model component is used to illustrate the approach and to prove, how it is based on a theoretical foundation. The use of the modelling approach is illustrated by presenting some applications related to an engineer to order company, which must accept long order horizons and many changes of the orders both before and after order acceptance. Therefore, it is necessary to concentrate on decisions, which are relatively invariant throughout order processing. By use of the presented modelling approach, it is shown, how modelling on multiple abstraction levels can be a solution to such challenges.

Keywords

Mass customisation, product configuration, product family modelling, information modelling, classification, composition, object-oriented analysis and design.

Introduction

Mass Customisation (MC) was initiated more than one decade ago as a research topic with Davis' publication "From Future Perfect: Mass Customisation" [Davis, 1989], presenting how products and services could be realised as a one-of-a-kind manufacture on a large scale. Davis also presented the idea that the customisation could be done at various points in the supply chain. In 1993, Pine published a major contribution to the mass customisation literature: "Mass Customization: The new Frontier in Business Competition" [Pine, 1993], [Pine et al., 1993], which was an extensive study of how American enterprises during the seventies and eighties had been overrun by the efficient Japanese manufacturers, which could produce at lower costs and higher quality. Since its introduction, MC has called for a change of paradigm in manufacturing and several companies have recognised the need for mass customisation. Much effort has been put into identifying, which success factors are critical for an MC implementation and how different types of companies may benefit from it [Lampel and Mintzberg, 1996], [Gilmore and Pine, 1997], [Sabin, 1998], [Berman, 2002], [Silveira et al., 2001].

For obvious reasons, there are different strategies on how to implement MC most appropriately and it varies naturally also between different companies, markets and products. Because there is not a single generic strategy, it is important to look at the issue from different viewpoints. The fact that products must be easily customisable in order to achieve MC has been described comprehensively in the literature. [Berman, 2002] and [Pine, 1993] proposed that the use of modular product design combined with postponement of product differentiation would be an enabler to a successful MC implementation. This issue of course also relates to the question of readiness of the value chain.

Mass Customisation and Product Configuration

An often used approach for implementation of MC is *product configuration*, in which a series of products is defined by one single model – a product family model (see figure 1) [Jørgensen, 2003]. Hence, a *product family* can be viewed as the set end products, which can be formed from a product family model. In the product family model, it is described, which modules are included in the product family model and how they can be combined [Faltings, 1998]. The result of each configuration will be a model of the configured product, *configured product model*. From this model, the physical product can be produced (see figure 1).

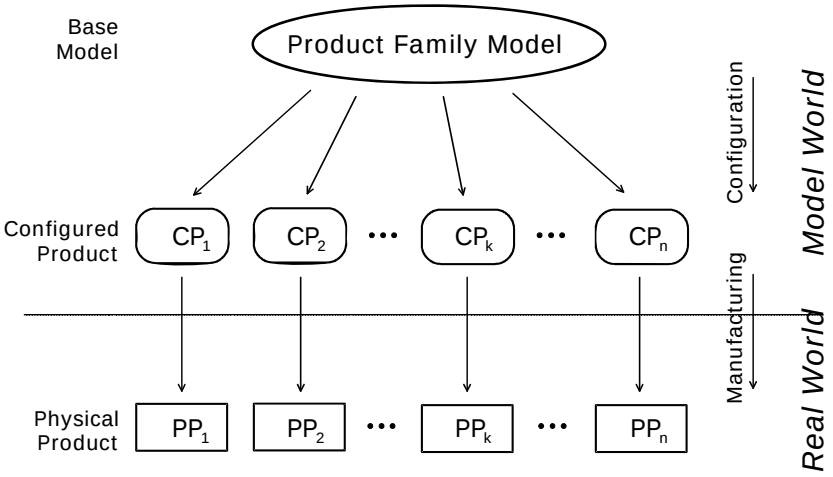


Figure 1 – Product Family Model as basis for configuration

From time to time, several different methods for defining product family models and product configurators have been proposed, each with their own advantages. A "Procedure for building product models" is described in [Hvam, 1999] based on [Hvam, 1994]. It is a rather practical approach with a seven step procedure, describing how to build a configuration system from process analysis and product analysis onto implementation and maintenance. For the product modelling purpose it uses the Product Variant Master method followed by object-oriented modelling to describe both classification and composition in a product family. The object-oriented approach is also applied by [Felfernig et al., 2001], who uses the Unified Modelling Language (UML) to describe a product family. This is done by using a UML meta model architecture, which can be automatically translated into an executable logical architecture. In contrast to [Hvam, 1999] this method focuses more on formulating the object-oriented product structure, rules and constraints most efficiently. The method also focuses on how the customers' functional requirements can be translated into a selection of specific modules in the product family.

Most of the methods, which exist for product family modelling, focus on modelling of the solution space of a configuration process. This means that they describe the possible attributes of the products and the product structure. Hence they do typically not focus on additional information which goes beyond, what must be used to perform the configuration itself. This kind of information, which could include e.g. customer, market, logistics and manufacturing information, is according to [Reichwald et al., 2000] similarly important, since a successful implementation of MC must integrate all information flows in the so called "Information Cycle of Mass Customisation".

In [Jiao et al., 1998], [Du et al., 2000] and [Männistö, 2001], mapping of functional requirements to specific modules is considered. Jiao proposes to use a triple-view representation scheme to describe a product family. The three views are the functional, the technical and structural view. The functional view is used to describe, typically the customers, functional requirements and the technical view is used to describe the design parameters in the physical domain. The structural view is used for performing the mapping between the functional and technical view as well as describing the rules of how a product may be configured. The description of this modelling approach is however rather conceptual, and does not easily implement in common configuration tools.

A product family model is often the basis for development of a *product configurator,* a tool, computer software, which can support users in the configuration process [Faltings, 1998], for instance by selecting modules to compose products. Hence, with a product configurator, it is possible to configure multiple individual solutions – perhaps a large set of products.

Product Family Models

A product family model (see figure 1) [Jørgensen, 2003] has a set of open specifications, which have to be decided to determine or configure an individual product in the family. The product family model is a foundation for configuration and, in order to secure that only legal configurations can be selected, the family model should contain restrictions about what is feasible and not feasible. Hence, the product family is defined as the set of possible products, which satisfy the specifications of the product family model. The result of each configuration will be a model of the configured product. From this model, the physical product can be produced (see figure 1). A *product configurator* can be defined as a tool, computer software, which is built on the basis of a product family model and which can support users in the configuration process [Faltings, 1998].

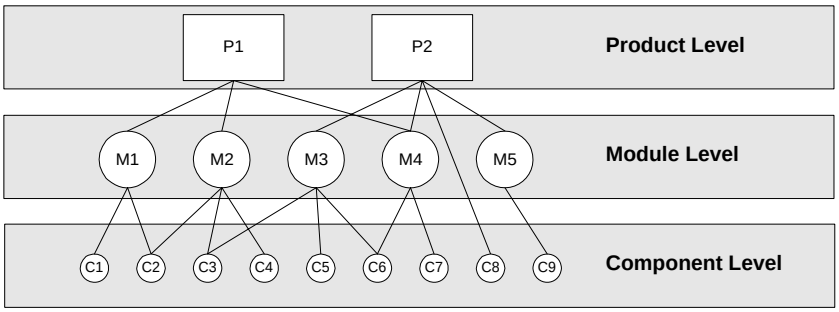


Figure 2 *Model of the structure with the three levels.*

Product configuration in the simplest form is a matter of combining a set of *modules* (see figure 2) so that the product model contains information about what modules and components are to be assembled. In this *compositional view,* a product consists of a number of components, which subsequently can consist of other components, etc. Modules are identified on a level above components from a configuration point of view whereas components usually are identified from a manufacturing point of view. In general, modules can be defined on multiple levels and can be configurable too. Most often, the number of modules is smaller than the number of related components. Thus, in the *structural model* for configurable products, products consist of modules and modules consist of components.

In connection with identification of modules, it is important to analyse how modules interface with each other. Therefore, it is important also to look at the modules functional characteristics and secure that the modular structure is harmonised with the functional division of the product [Andreasen, 2003].

Besides structure, products have *properties/attributes*. It is essential for both the customer and the producer to focus on properties/attributes of the resulting product. For each configured product, the resulting properties are dependent of the selected components and structure of the product.

The dependencies between properties/attributes and module structure is illustrated in figure 3, which shows how underlying modules/components are determined on the basis of decisions regarding the chosen attributes. Five different alternatives are shown of which alternative number 1 is an exception from the general scheme.

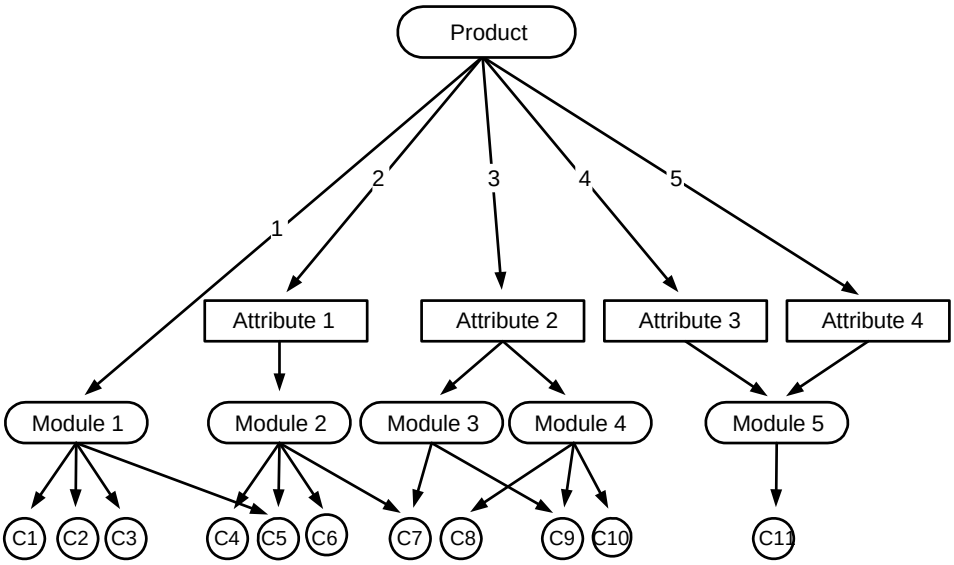


Figure 3 Specification of modules directly or indirectly through functionalities.

At selection no. 1, a specific attribute is not selected, because this is a case, where it is more natural to choose a module directly - typically *add-on modules*. An example of this is the sunroof of a car (provided that only one type of sunroof exists). Simply: Sunroof (Yes/No?).

At selection no. 2, attribute1 is equal to module 2. This can only be fulfilled in one way, and that is by including module no. 2. For instance, air conditioning equals an air condition module.

At selection no. 3, attribute no. 2 results in that both module no. 3 and 4 are selected. An example of this is that if the customer chooses the turbo car model, then both a turbo engine and ABS brakes must be selected.

Finally, selections no. 4 and 5 show a relatively usual case, where a module is determined by more than one attribute, i.e. attributes of the module. For instance, a seat can be specified from two attributes: the colour and whether there should be a headrest or not. When these two attributes are specified, then one module (a complete seat) can comply with both attributes.

Mapping of functional requirements to specific modules is considered in [Jiao et al., 1998], [Du et al., 2000] and [Männistö, 2001]. Jiao proposes to use a triple-view representation scheme. The three views are the functional, the technical and structural view. The functional view is used to describe, typically the customer's functional requirements and the technical view is used to describe the design parameters in the physical domain. The structural view, which corresponds to the structural level described above, includes the mapping between the functional and technical view as well as the rules of how a product may be configured. The description of this modelling approach is however rather conceptual, and does not easily implement in common configuration tools.

In the product configuration process, algorithms must be available to estimate the resulting product properties. Some properties are simply the properties of the components, e.g. the colour of a car is normally defined as the colour of the car body. Other properties are computed from properties of the components. For example, the weight is simply the sum of the component's weight. However, not all resulting properties are so easy to determine. For instance, the resulting performance of a pump is a non-linear function of certain component properties. Much more complicated examples could be mentioned.

Contrasting to this, inverse algorithms must also exist in the case, where the structure is determined by attributes as described along with figure 3. If, for example, pumps are to be configured by requirements to performance attributes like pressure and flow, the algorithm must be able to determine which pump configurations can satisfy these requirements. Such algorithms will often be rather complicated to develop. Often, an analytic solution can not be developed with reasonable resources. Instead, a search algorithm may be easier to develop. If the performance attributes are known and listed in tables or can be calculated for each possible configuration, the algorithm can perform a search through all these options. Typically, requirements to the performance attributes are combined with constraints like "best fit" or "cheapest fit". I such cases, an optimisation problem may be formulated.

In the following, the term *attribute* will be used in the models corresponding to properties of physical products. Consequently, when a configuration is performed, the desired properties of the resulting product must be determined by defining values of attributes in the product family model. All relevant attributes of both the resulting product and the available modules must be specified and their optional values to be selected during configuration tasks must also be defined. In relation to this, it is important to notice that the selectable modules and components are sometimes substituted by one or more attributes. For instance, a computer can be ready for use (attribute) or the operating system is installed (module/component). Therefore, the configuration process can be considered as a mixture of attribute specification and selection of modules, which together can satisfy the required attribute values.

Fundamental Issues of Information Modelling

Methodologies for system development are often based on concepts derived from General Systems Theory. According to this theory, a *system model* is an intentionally simplified description of a system, fulfilling a certain purpose. Hence, the simplifications imply that some choices are made in order to select the most important properties, components and relationships. Thus, a system model can e.g. be suitable for communication between designers, because with the model, it will be possible to concentrate on the most important aspects of the system. Models are viewed either as *analysis models* or *synthesis models.* Analysis models are models of something existing, often physical objects and synthesis models are models created as a foundation for construction of something new, which eventually will become physical – an artefact [Jørgensen, 2002]. Hence, synthesis models are built purely from ideas, thoughts and imaginations and obtained in some kind of representation. Design by modelling is a development approach, where a synthesis model is designed as an intermediate result and the final result is an implementation of the model in the real world.

Computer-based models are fundamentally stored in computers as *data objects* and *data structures*, which can be manipulated by applications. Therefore, development of tools for modelling includes both development of a *data model* and a number of *applications* with relationships to the data model [Jazayeri, 2000]. One of the most important requirements for the data model is that it is non-redundant so that no data value is stored more than once. In order to ensure that this requirement is fulfilled, the model representation has to be considered very carefully based on the meaning of data, the semantics. Therefore, the foundation for a data model is an *information model* ([Hammer 1978], [Rumbaugh et al. 1999] and [Halpin 2001]), created in combination with semantics from the domain, which the design model is addressing.

An important fundamental issue of information modelling is *abstraction mechanisms*, which provide the means for identification and design of invariant components and structures ([Smith 1977a], [Smith 1977b], [Rosch 1978] and [Sowa 1984]). Two abstraction mechanisms are defined here: *composition* and *classification* [Jørgensen, 1998]. Composition focuses on the components and the relationships between the components. The most frequently used structure is the component structure, which shows *aggregation* versus *separation*. Such a structure is illustrated in figure 4 for a sample computer.

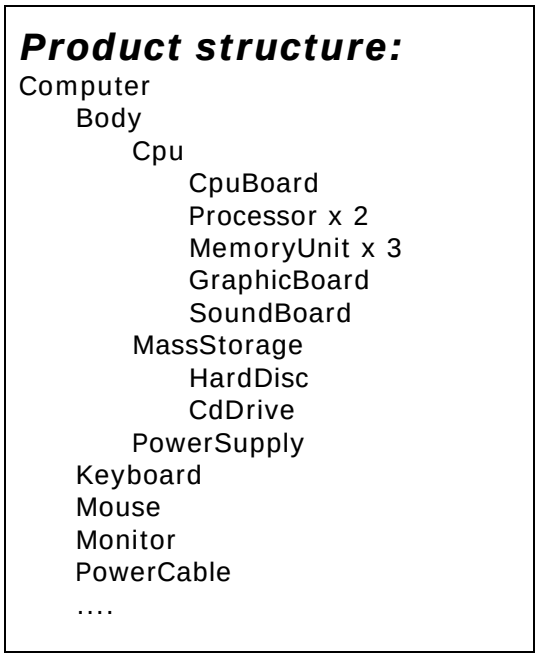


Figure 4 – Sample composition structure of a computer

Classification focuses on identification of *classes/types* of components based on the *properties/attributes*, which characterise them. This can be illustrated in a diagram, termed *taxonomy* (see figure 5), where the relationships *generalisation* versus *specialisation* are shown. Often, a UML class diagram is used for the taxonomy ([Rumbaugh et al. 1999]).

In information modelling, composition and classification together support identification of fundamental structures on a *type level* as the basis for generation of individual components on an *instance level* and they provide the means to set particular focus on the most invariant decisions. A classification process results in a basic structure of types and a composition process results in a basic structure of components.

Another important issue of information modelling is the *object-oriented paradigm,* which can be adopted in harmony with the abstraction mechanisms. In this paradigm, each model component is regarded as a living organism, which act and interact with other components. Thus, object-oriented components are equipped with behavioural attributes, which enable them to respond to requests and, consequently, even if a real world component is non-living, the corresponding model is created as an active component.

Taxonomy:

Computer components
- Mass storage components
  - Hard discs
  - Cd drives
  - Dvd drives
- Print boards
  - Cpu boards
  - Graphic boards
  - Io boards
  - Sound boards
  - Tv tuner boards
- Integrated circuits
  - Processors
  - Memory Units
- Cpu modules
- Mass storage modules
- Cables
  - Power cables
  - Disc cables
- Other
  - Bodies
  - Power supplies
  - Keyboards
  - Mice
  - Monitors
  - Computers
- ....

Figure 5 – Sample taxonomy of computer components

The two abstraction mechanisms are used in design tasks, but, as indicated in figure 6, classification is used first and composition afterwards. Classification primarily supports the identification of model components and the basic structure at the type level. Based on this, the structural considerations are identified by use of composition.

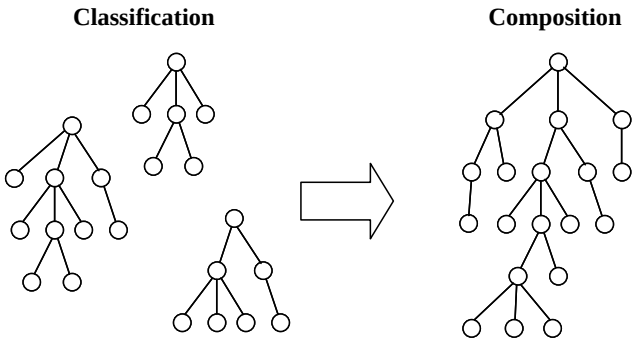


Figure 6 – Classification and composition hierarchies

A Generic Information Model Component

In order to be able to create all sorts of models and to perform many different modelling processes, a conception of a *generic model component* is introduced. This component is inspired from general systems theory and from object-oriented modelling and can be regarded as a component that can be used for system models in general and for information modelling.

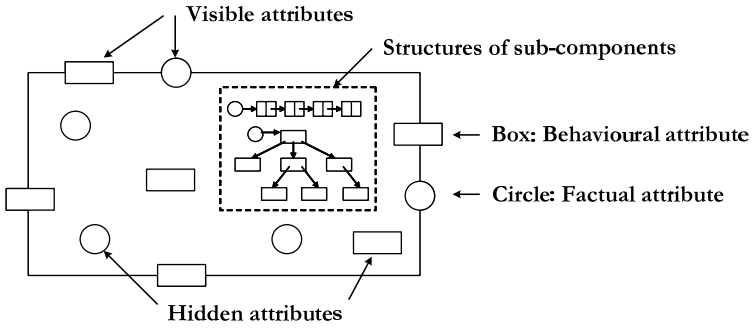


Figure 7 - Generic model component

The generic component consists of a set of *attributes* and a *structure of sub-components* (see figure 7). Some attributes are *factual attributes*, defining the state of the component, and some attributes are *behavioural attributes*, defining the operations, which the component can carry out. An alternative division of attributes defines some attributes as *visible attributes*, which can be called from other components, and some are defined as *hidden attributes*. The structure establishes the relationships between the component itself and the sub-components.

All structures can be represented by two kinds of relationships in the information model: *references* and *collections*. For the computer example, a reference could represent the relationship e.g. between the keyboard and the computer. A collection could represent the relationship e.g. between the cpu board, the anchor, and multiple memory units, the members.

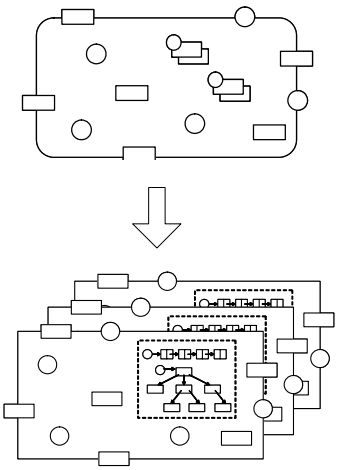


Figure 8 - Object type is the basis for generating objects (instances)

When a synthesis information model is considered, a foundation for the components must be established by creating *types* of components. Component types are the primary content of information models and it is important to distinguish between modelling on the object level and modelling on the type level.

Each component type includes a specification of a set of attributes with *name* and *data type*. The classification abstraction mechanism is primary because, based on attributes, the component types can be classified and organised in a hierarchy, the taxonomy. Identification and specification of structures can also be included in the component types by creating the *relations,* which formulate the *constraints* regarding attributes and combinations of sub-components. The component type is a kind of template and, from each type, an indefinite number of components, instances, can be generated. The quality of these component types is the key basis to achieve an invariant information model foundation.

Product Family Modelling

There is a need for a methodology to describe and develop models of configurable products. Companies, who are implementing product configuration, need a comprehensive terminology and a systematic methodology in order to develop their modular products. It is of great importance to use well-defined terms and use the agreed terminology consistently in connection with a well-proven methodology, so that misunderstandings can be avoided and communication can be eased.

Attributes and Data Structures

As mentioned, products consist of properties, components and structure and similar contents goes for models of products and product families. In the following, the term *attribute* will be used in the models corresponding to properties of physical products. Consequently, when a configuration is performed, the desired properties of the resulting product must be determined by defining values of attributes in the product family model. All relevant attributes of both the resulting product and the available modules must be specified and their optional values to be selected during configuration tasks must also be defined. In relation to this, it is important to notice that the selectable modules and components are sometimes substituted by one or more attributes. For instance, a door can be lockable (attribute) or it can be equipped with a lock (module/component). Therefore, the configuration process can be considered as a mixture of attribute specification and selection of modules, which together can satisfy the required attribute values.

Development of Product Family Models

As stated above, product family models must be able to construct individual product models through a configuration task. Each product model must have sufficient data about attributes and structure to describe and manufacture the physical product. Consequently, the basic elements of product family models are the total set of attributes of the possible product models and the set of identified modules, each with their internal attributes and data structures.

The basic units of a product family model are module types. A *module type* is a model of the set of modules, which are interchangeable, perhaps with some restrictions. During

configuration, individual modules of each type are determined. The attributes of the product models and the module types are selected on the basis of what is important and relevant.

In the following, the contents of product family models are illustrated by use of simple elements of a synthetic language. Furthermore, fractions of a simple example of a computers product family model are added to the illustration.

Each attribute in a module type is defined by a *name* and probably a *data type* (Boolean, Integer, Float, String, Currency, etc.).

This declarative statement shows the syntax for description of a module type:

type name {…}

Example:

```
type HardDisk {...}
```

The syntax of attribute declaration with data type is:

name : data type;

Example:

```
type HardDisk
{
    Name :              string(50);
    StorageCapacity :   integer;
    AccessTime :        float;
    Price :             currency;
}
```

The available instances of a module type can be listed by a table with a column for each attribute and a row for each module.

Name	*StorageCapacity*	*AccessTime*	*Price*
Maxtor 10K-3	37 Gb	4,5 ms	1.375 DKK
Maxtor 10K-4	147 Gb	4,4 ms	4.055 DKK
Maxtor 10K-5	300 Gb	4,4 ms	8.975 DKK

Alternatively, module data can be extracted from a database.

Some modules can be configured by selecting attribute values. In this case, each attribute is not defined by a data type but instead by a *domain* with the possible values. A domain can be a set of discrete values, an interval of integer values or a list of named values.

The syntax of an attribute declaration with domain and a possible default value is:

name : {domain} [default value];

Example:

```
type HardDisk
{
    .....
    PreSet :            {Master, Slave} default Master;
    OperatingSystem :   {Non, WinXP, Win2000, WinMe} default WinXP;
    .....
}
```

When module data are specified in form of a table as shown above, the selection of domain values can be added as columns to the table.

Attributes of a module can be a function of other attributes in the same module or in other modules. This can be modelled by an *expression* with standard functions or special functions as a special algorithm. If the name of a module type is included in such an expression, it means “number of instances of the type”.

Examples:

```
type Computer
{
    OperatingSystem : Boolean default true;
    Colour =          Case.Colour;
    HardDisks =       HardDisk;
    DiskMemory =      Sum(HardDisk.StorageCapacity);
    Weight =          SumWeight : Double { ... Specific algorithm ... }
    .....
}
```

Structures are represented by special kind of attributes.

The symbol -> represents a *reference* i.e. a one-to-many relationship

Example:

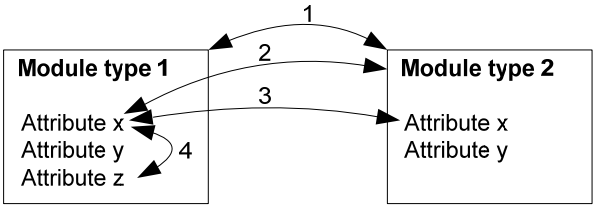
```
type Processor
{
Name : string;
...
ContainingBoard -> CpuBoard;
}
```

The symbol ->> represents a *collection* i.e. a one-to-many relationship

Example:

```
type Cpu
{
...
RelatedGraphicBoards ->> GraphicBoard;
RelatedIoBoards ->> IoBoard;
...
}
```

Typically for module types, it is possible to add *relations*. In general, there are four different kinds of relations, see figure 9.



Explanation:

1: Constraints between module types

2: Constraints between module types og attributes

3: Constraints between attributes in different module types

4: Internal constraints between attributes in the same module type

Figure 9 – Four kinds of relations

Among other things, relations of category 1 are used to specify product structures. Here, it is described that a product/module (instance of a module type) consists of modules (instances of other module types), which eventually also consist of modules etc. until the component level is reached. In a module type, such a relation expresses the module types for possible sub modules. Furthermore, a *multiplicity* is specified in order to form a basic expression about the number of instances that can be included.

The syntax for relations describing contents is:

contents { multiplicity module type; ... }

Multiplicities is formulated with the syntax:

from .. to

where from is typically 0 or 1 and to is typically a fixed number or any number greater than or equal to from. This is indicated by a *.

Examples:

1.. from one to many*
0.. from none to many*
1..1 one and only one

Example:

```
type Cpu
{
    .....
    contents { 1..1 CpuBoard;  1..* Processor;  1..* MemoryUnit ;}
    .....
}
```

```
type ComputerCase
{
    .....
    contents  { 1..1 PowerSupply;   0..* PowerCable; }
    .....
}
```

All other kinds of relations are formulated by arithmetic expressions. Here, the ordinary arithmetic operators like addition, subtraction, multiplication and division can be used together with standard functions. The following arithmetic relation operators =, >=, <=, >, < and <> can also be used along with the logical operators AND, OR, XOR, NOT, implication (⇒) and bi-implication (⇔). If the name of a module type is included in a logical expression, it means ”instance of the type”.

Examples of relations with arithmetic and logical operators are:

```
type Cpu
{
    constraints
    {
        GraphicBoard + IoBoard + TvTunerBoard <= NbOfBusSlots;
        Processor <= ProcessorSlots;
        .....
    }
}

type Computer
{
    constraints
    {
        Monitor <= 2;
        HardDisk + CdDrive + DvdDrive  <=  DiskCable * 2;
        OperatingSystem ⇒ HardDisk.OperatingSystem <> Non;
        CdDrive not ⇔ DvdDrive;
        .....
    }
}
```

As previously stated, the classification abstraction mechanism is fundamental for identification and definition of types; hence, the module types above are actually related to each other as indicated in figure 5.

The syntax of the relationships between super-types and sub-types is:

type *name1* subtypeof *name2* { ... }

Examples:

```
type ComputerComponent { ... }

    type MassStorageComponent subtypeof ComputerComponent { ... }

        type HardDisk subtypeof MassStorageComponent { ... }

    type Cpu subtypeof ComputerComponent { ... }
```

type IntegratedCircuit subtypeof ComputerComponent { ... }

type Processor subtypeof IntegratedCircuit { ... }

type Other subtypeof ComputerComponent { ... }

type Computer subtypeof Other { ... }

With classification, it is defined that attributes in super-types are inherited to sub-types.

Abstraction by Classification

Regardless of whether the selection of modules is implicit or explicit, multiple abstraction levels can also be established by the use of classification. In a taxonomy over module types (see figure 5), the types towards the root are the most general types whereas the types towards the leaves are the most special types. Therefore, a selection of relatively general types represents a higher abstraction level compared to selection of relatively special types.

Taxonomy:
Computer components
- ...
- Print boards
  - ...
  - Sound boards
    - Surround
      - 4.1 channels
      - 5.1 channels
      - 6.1 channels
    - Stereo
      - Ordinary
      - Four point
    - 3D
    - ...
- ...

Figure 10 – Further classification of sound boards

Figure 10 shows a partial taxonomy as a further classification of a specific module type of figure 5 and reveals two additional levels of specialisation. Clearly, this example illustrates that a preliminary selection of a relatively general type is a way of postponement, i.e. some indications are given but further specifications can be submitted.

All module types have attributes, which can be included in the configuration process. Besides an obvious price attribute, further technical properties of the available modules can be represented as attributes of the module types. These attributes can be located at different levels of the taxonomy depending on how general or special they are. Consequently, a selection of a type results in a set of additional attributes, which can be used for further specification. However, if a specification of a specific attribute is required, a specialisation down to a certain level is implicitly made. If for instance something is required

about attributes which are only relevant for stereo sound, then stereo sound boards are implicitly selected.

In general, classification is very much related to attributes. Besides what is already described, identification of sub-modules can be based on values of attributes. For instance, the sub-types of surround sound board could be identified by values of an attribute 'NoOfChannels'. In fact, this attribute could remove the need for classification at the lowest level. Hence, if multiple classifications of these sound boards were relevant, i.e. if multiple and equally important classification criteria exit, it will be more flexible to identify the corresponding attributes and their possible values.

Sample Applications of Product Family Modelling

Many observations indicate that implementation of Mass Customisation and product configuration in ETO companies must focus on product modelling in order to gain immediate economic results from saving resources for tendering and order processing. This top-down development approach is also important when different organisational units must be joined and different software applications and databases must be integrated. Therefore, a number of theoretical topics about system modelling, product modelling, modelling of product families, information modelling and data modelling must be utilised.

In this paper, it is proposed that modelling of product families should be performed in a way that multiple levels of abstraction can be identified and a top-down configuration approach with specification of attributes and structure. This is especially suitable for order processing over long time, where it is important to control the degree of freedom at different steps. It is necessary to postpone certain decision until enough requirements are available.

The proposed approach is currently under implementation at the Danish case company, Aalborg Industries, which is producing a range of boiler plants, primarily for the marine sector. Here, the development of product family models and product configurators has been carried on for several years starting with a simple model for calculation of quotations. In later versions, data from the product configurator has been used as parameter input to other software applications for producing data sheets and drawings. This development has proved the necessity to set greater focus on product modelling on multiple abstraction levels.

The current version of the product configurator is web-based so that sales and tendering can take place everywhere around the world. This technology will also be used in the future and the company is now developing a more advanced product model and related product configurator software modules with the purpose of integrating more of the existing software applications and get more optimised order processing and production planning. Furthermore, supply chain management issues are taken into consideration so that decisions about selection of manufacturing locations and suppliers can be optimised. Especially, issues about interaction with ERP systems are important and require software modules for automatic interfacing.

As described for the case company, the order horizon can be rather long and many changes in the order specification occur. Hence, for this company, it will be important to rise to a higher abstraction level by setting focus on specification of attributes and move away from the structural model of configuration.

Configuration by performance

When configuring a boiler plant, the main functional attributes are often the following: 'Steam capacity', the amount of steam which the boiler is able to produce, 'Working pressure', the pressure of the steam delivered by the boiler, 'Waste heat recovery capability', the ability of the boiler to transform heat from exhaust gas to steam and 'Fuel oil type', the type and viscosity of fuel oil the boiler can use.

The 'Steam capacity' and 'Working pressure' are the main attributes that determine which type of boiler will meet the customer's requirements. Although some combinations of capacity and pressure will allow the selection of different product types, the configuration of these parameters will delimit the possible product types. Values of these parameters will also have a major impact on other physical form of the boiler such as geometry and dry weight. The waste heat recovery capability of the boiler however, is a binary attribute and the selection of this will have major influence and narrow down the solution space. Finally, the fuel oil type may also constrain the solutions substantially. If for example heavy fuel oil is to be used, a number of product types will not be an option, since these are not constructed to be able to burn heavy fuel oil, but solely marine diesel. The selection of fuel type will also have a direct implication on the burner selection, which is a sub module to the boiler. Hence, the configuration of this parameter will have implication on both the boiler, and the subcomponents of the boiler.

Say for instance that a customer requires a 'Steam capacity' of 20 ton/hour, no 'Waste heat recovery capability' and 'Fuel oil type' is marine diesel. Then the solution space is reduced to two product families 'Mission OM' and 'Mission OL'. As 'Mission OL' is the cheapest for equal capacities, this will be the primary choice. However 'Mission OL' is higher than 'Mission OM' and must therefore be selected if the height is limited. Furthermore, multiple burners are available for 'Mission OM' and not for 'Mission OL'. If, in addition, a certain 'Working pressure' is needed the thickness of the boiler shell must be set appropriately. A pressure of 9 bar requires 18 mm thickness, while 18 bar increase the thickness to 32 mm.

As indicated, it will be very difficult to develop an algorithm, which can determine the structure based upon requirements to the functional attributes. Consequently, the preliminary plan is to develop a search algorithm, which can scan the possible solutions. This algorithm will most likely also need to handle additional constraints like 'Cheapest fit' or '3 best fit' and it must necessarily also provide some cost data.

Module supply

Many modules can be purchased as products from multiple suppliers, which can deliver with a variety of properties for sizes, price, performance, quality, lead time, etc. Two examples from the case company can illustrate this. In the first example, alternative feed water pumps for boilers can be selected as illustrated in table 1.

	Delivery head Bar(gauge)	Capacity m^3/h	Supply voltage V	Price €
(Requirements)	(>= 22)	(>= 25)	(3 x 330)	
Product 1	23	25.5	3 x 330	1600
Product 2 *)	25	30	3 x 330	2000
Product 3 **)	24	25	3 x 330	1800

*) Has frequency converter drive, i.e. significantly lower power consumption
**) Approved for running in explosion risky zones

Table 1 – Alternative feed water pumps specified with a set of attributes

The table shows that three sample requirements are specified and that tree different pump products can satisfy the requirements. It also shows that additional attributes may be taken into consideration if further specifications have to be made.

In the second example, it is shown that alternative safety valves can be selected (see table 2).

	Set pressure Bar(gauge)	*Size*	*Production location*	*Delivery time*	*Price €*
(Requirements)	(19)	(DN50)	(Deliv. location: Finland)		
Product 1	19	DN50	Germany	2 days	200
Product 2	19	DN50	China	30 days	130

Table 2 – Alternative feed water pumps specified with a set of attributes

Two valve products satisfy the requirements but, as shown, with great difference between the prices. A significant attribute is the delivery time, which may set serious limitations regarding the time for procurement. However, this is dependent on the production location so, if for instance the production location is changed to the East Asia, a dramatic reduction of delivery time and price can be reached.

Abstraction by Classification

Two examples of abstraction by classification can also be presented (see [Jørgensen, 2008] for description of the syntax). Example one is about oil fired boilers, where the module type 'OilfiredBoiler' is the super-type for two sub-types 'MissionOS' and 'MissionOL'. Two attributes show the decision making, 'BurnerType' and 'Capacity'.

```
type OilfiredBoiler
{
    BurnerType :    {KB,KBO,KBE,KBSA,KBSD};
    Capacity :      [1.6 .. 15.5];
}

type MissionOS subtypeof OilfiredBoiler
{
    BurnerType :    {KB,KBO} default KB;
    Capacity :      [1.6 .. 6.5];
    ...
}

type MissionOL subtypeof OilfiredBoiler {...}
Etc.
```

For oil fired boilers, the burner type can be any of the listed values, while for mission OS boilers only a subset of burners is valid. The capacity for mission OS boilers is similarly narrowed compared to the oil fired boilers in total.

Example two regards feed water pump units, where there are two sub-types and where the regulation type differs.

```
type FeedWaterPumpUnit
{    RegulationType :      {OnOff,Modulating}; }

type FeedWaterPumpUnitOnOff subtypeof FeedWaterPumpUnit
{    RegulationType :      {OnOff}; }

type FeedWaterPumpUnitModulating subtypeof FeedWaterPumpUnit
{    RegulationType :      {Modulating}; }
```

Both examples show that the super-type modules represent decisions on a higher abstraction level because selection of a general module type establish some degree of specification while remaining decisions are postponed. In contrast, sub-types represent decisions about more precise specifications. In the sales process, it will be possible to assist the customers with decisions about how specific they must be from the beginning. A balance must be obtained. Relatively specific decisions give more precise estimations (cost, required capacity, delivery, etc.) but are most likely subject to changes and, on the other hand, decisions on a more general level will lead to uncertainty about estimations. A key issue in relationship with configuration is to develop models for calculating estimations based on different levels of abstraction in decision making.

Conclusion

In this paper, it is underlined that there are some fundamental issues of information modelling, which can be applied to product family modelling. For Product family models, it is important to identify the attributes in the model of the end-products and, because some attributes in models of product families will be assigned values during the configuration process, they must be defined with optional values i.e. domains. It is also characteristic for product family models that relations/constraints must be defined between attributes of the possible end-products and the attributes of the identified modules/components.

As a basis for development of detailed information models, a generic model component is presented. Likewise, a generic component type is introduced as the basis for creation of information models. According to this type, the basic content of product family models is proposed in form of a module type and a simple synthetic language is presented. The use of this module type is illustrated by a number of examples.

In the paper, special focus is set on how to develop product family models, which can support product configuration on multiple abstraction levels – suitable for some engineer-to-order companies with long order horizons. First of all, it is proposed that configuration is performed by specification of attributes instead of selection of modules. This means that the structure of end-products is defined indirectly based on the values of attributes. Thereby, configuration is more oriented towards customer needs because attributes are essential in connection with the functional demands from customers. Further, it is proposed that, when modules are selected, it is important to develop classifications of module types and form a taxonomy. Such a structure is well suitable for identification of multiple abstraction levels by classification, where specifications can range from a general level to a more specific level.

The aim of developing product family models is that they can be used as a foundation for development of specific product configurator software and the proposed methodology, included in this paper, is for the moment being used by a particular ETO company, which intend to develop an advanced product family model and a product configurator that can support many organisational functions in the company world wide. Especially, the top-down approach with modelling on multiple abstraction levels are followed very closely and considerable amount of specially designed software modules are being developed.

References

[Andreasen 2003]
Andreasen, Mogens Myrup (2003). Relations between modularisation and product structuring. In Proceedings of the 6th workshop on Product Structuring – application of product models, MEK-DTU, Denmark, pp. 1-15.

[Berman 2002]
Berman, B. (2002). Should your firm adopt a mass customization strategy? Business Horizons, 45(4):51–60.

[Davis 1989]
Davis, S. (1989). From future perfect: Mass customizing. Planning Review.

[Du et al. 2000]
Du, X., Jiao, J., and Tseng, M. M. (2000). Architecture of product family for mass customization. In Proceedings of the 2000 IEEE International Conference on Management of Innovation and Technology.

[Faltings 1998]
Faltings, Boi and Freuder, Eugene C. (Ed.): *Configuration - Getting it right*. Special issue of *IEEE Intelligent Systems*. Vol.13, No. 4, July/August 1998.

[Felfernig et al. 2001]
Felfernig, A., Friedrich, G., and Jannach, D. (2001). Conceptual modeling for configuration of mass-customizable products. Artificial Intelligence in Engineering, 15:165–176.

[Gilmore and Pine 1997]
Gilmore, J. and Pine, J. (1997). *The four faces of mass customization.* Harvard Business Review 75 (1).

[Halpin 2001]
T. Halpin: Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design, 3rd ed. Morgan Kaufmann 2001.

[Hammer 1978]
Hammer, Michael and McLeod, Dennis: *The Semantic Data Model: A Modelling Mechanism for Data Base Applications.* Proceedings of ACM/SIGMOD International Conference on Management of Data. Austin Texas, pp.144-156, 1978.

[Hvam 1994]
Hvam, L. (1994). Anvendelse af produktmodellering, -set ud fra en arbejdsforberedelsessynsvinkel. PhD thesis, Driftteknisk Institut, DTU.

[Hvam 1999]
Hvam, L. (1999). A procedure for building product models. Robotics and Computer-Integrated Manufacturing, 15:77–87.

[Jazayeri 2000]
Jazayeri, M.; Ran, A. and van den Linden, F.: *Software architecture for product families: Principles and practice*. Addison-Wesley, 2000.

[Jiao et al. 1998]
Jiao, J.; Tseng, M. M.; Duffy, V. G. and Lin, F. (1998). Product family modeling for mass customization. Computers & Industrial Engineering, 35:495–198.

[Jørgensen 1998]
Jørgensen, Kaj A.: *Information Modelling: foundation, abstraction mechanisms and approach*. In: Journal of Intelligent Manufacturing, vol.9, no.6, 1998. Kluwer Academic Publishers, The Netherlands.

[Jørgensen 2002]
Jørgensen, Kaj A.: *A Selection of System Concepts.* Aalborg University, Department of Production, 2002.

[Jørgensen 2003]
Kaj A. Jørgensen: *Information Models Representing Product Families*. Proceedings of 6th Workshop on Product Structuring, 23rd and 24th January 2003, Technical University of Denmark, Dept. of Mechanical Engineering.

[Lampel and Mintzberg 1996]
Lampel, J. and Mintzberg, H. (1996). Customizing customization. Sloan Management Review, 38:21–30.

[Männistö 2001]
Männistö, M. M.; Peltonen, H.; Soininen T. and Sulonen, R.: *Multiple Abstraction Levels in Modelling Product Structures.* Data and Knowledge Engineering no.36, pp.55-78, 2001.

[Pine 1993]
Pine, B. Joseph: *Mass Customization - The New Frontier in Business Competition.* Harvard Business School Press, Boston Massachusetts, 1993.

[Pine et al. 1993]
Pine, J., Victor, B., and Boyton, A. (1993). *Making mass customization work.* Harvard Business Review 71 (5), 71(5):108–119.

[Pine and Gilmore 1999]
Pine, J. and Gilmore, J. (1999). *The Experience Econmy : Work Is Theater & Every Business a Stage*.

[Reichwald et al. 2000]
Reichwald, R., Piller, F. T., and M¨oslein, K. (2000). Information as a critical succes factor or: Why even a customized shoe not always fits. In Proceedings Administrative Sciences Association of Canada, International Federation of Scholarly Associations of Management 2000 Conference.

[Rosch 1978]
Rosch, Eleanor: *Principles of Categorisation.* In: Cognition and Categorization. Laurence Erlbaum, Hillsdale, Jew Jersey, 1978.

[Rumbaugh et al. 1999]
Rumbaugh, J.; Jacobson, I. and Booch, G: *The Unified Modeling Language Reference Manual*. Addison-Wesley 1999.

[Sabin 1998]
Sabin, D. and Weigel, R.: Product Configuration Frameworks - A survey. In IEEE intelligent systems & their appplications, 13(4):42-49, 1998.

[Silveira et al. 2001]
Silveira, G. D., Borenstein, D., and Fogliatto, F. S. (2001). Mass customization: Literature review and research directions. Int. Journal of Production Economics, 72:1–13.

[Smith 1977a]
Smith, J. M. and Smith, D. C. P.: *Database Abstractions: Aggregation.* Communications of the ACM vol.20, no.6. pp.405-413 New York 1977.

[Smith 1977b]
Smith, J. M. and Smith, D. C. P.: *Database Abstractions: Aggregation and Generalization.* ACM transactions on Data Base Systems, vol.2, no.2. pp.105-133 New York 1977.

[Sowa 1984]
Sowa, John F.: *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, 1984.